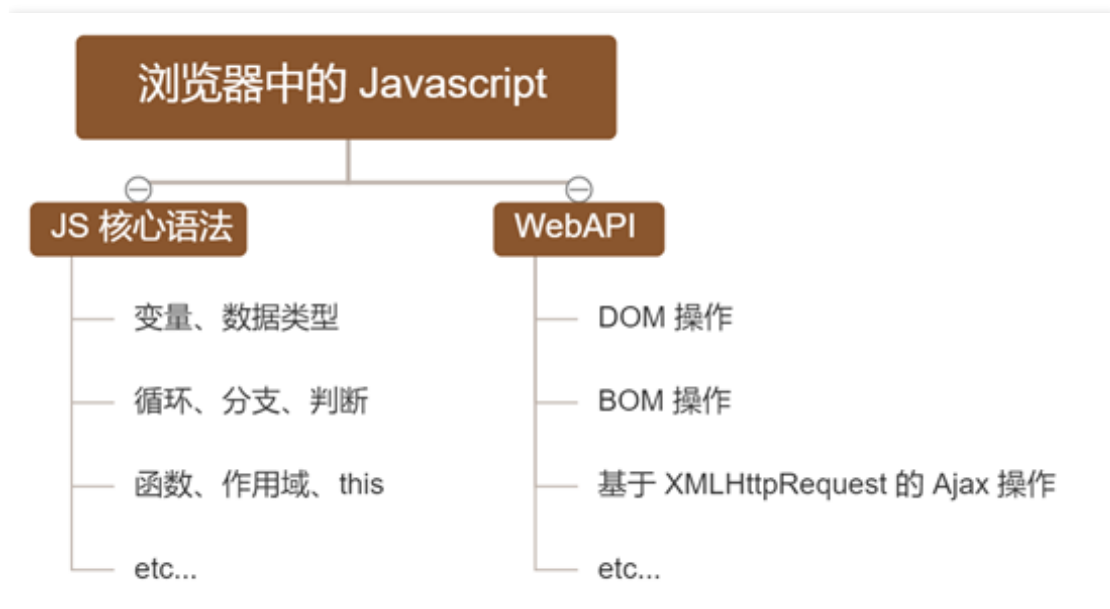


1、基础回顾

1.1、浏览器中的JavaScript的组成部分



1.2、为什么 JavaScript 可以在浏览器中被执行

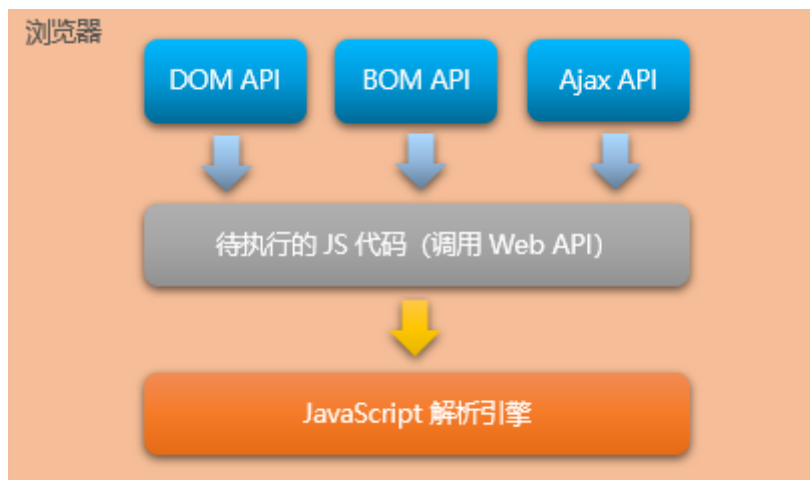


1.3、不同的浏览器使用不同的JavaScript解析引擎：

- Chrome 浏览器 => V8
- Firefox 浏览器 => OdinMonkey (奥丁猴)
- Safari 浏览器 => JSCore
- IE 浏览器 => Chakra (查克拉)
- etc...

其中，Chrome浏览器的V8解析引擎性能最好！

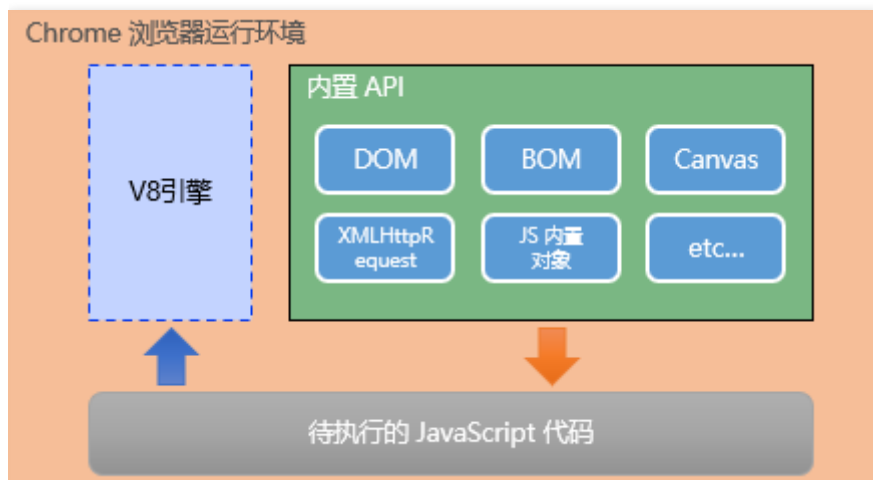
1.4、为什么 JavaScript 可以操作 DOM 和 BOM



每个浏览器都内置了 DOM、BOM 这样的 API 函数，因此，浏览器中的 JavaScript 才可以调用它们。

1.5. 浏览器中的 JavaScript 运行环境

运行环境是指代码正常运行所需的必要环境。



总结：

- V8 引擎负责解析和执行 JavaScript 代码。
- 内置 API 是由运行环境提供的特殊接口，只能在所属的运行环境中被调用。

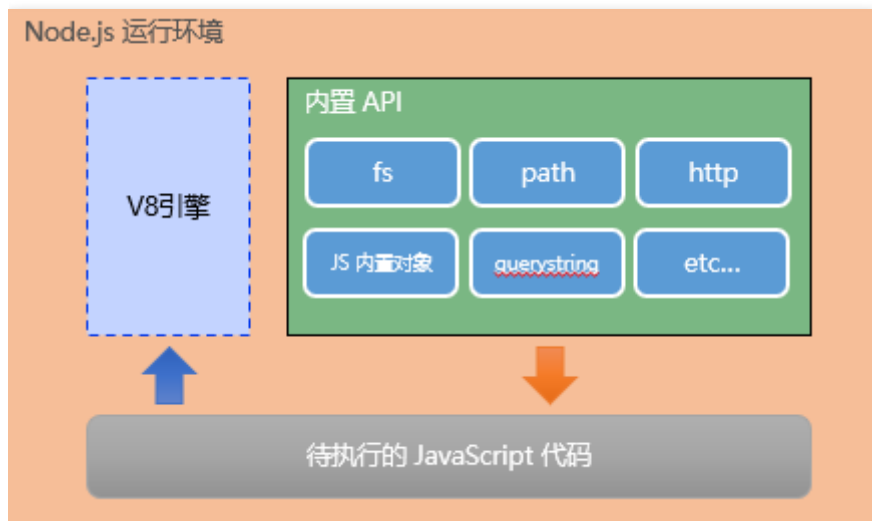
2、初识 Node.js

2.1、什么是 Node.js

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

Node.js 的官网地址：<https://nodejs.org/zh-cn/>

2.2、Node.js 中的 JavaScript 运行环境



注意：

- 浏览器是 JavaScript 的前端运行环境。
- Node.js 是 JavaScript 的后端运行环境。
- Node.js 中无法调用 DOM 和 BOM 等浏览器内置 API。

2.3、Node.js 可以做什么

Node.js 作为一个 JavaScript 的运行环境，仅仅提供了基础的功能和 API。然而，基于 Node.js 提供的这些基础功能，很多强大的工具和框架如雨后春笋，层出不穷，所以学会了 Node.js，可以让前端程序员胜任更多的工作和岗位：

- 基于 Express 框架 (<http://www.expressjs.com.cn/>)，可以快速构建 Web 应用
- 基于 Electron 框架 (<https://electronjs.org/>)，可以构建跨平台的桌面应用
- 基于 restify 框架 (<http://restify.com/>)，可以快速构建 API 接口项目
- 读写和操作数据库、创建实用的命令行工具辅助前端开发、etc...

总之：Node.js 是大前端时代的“大宝剑”，有了 Node.js 这个超级 buff 的加持，前端程序员的行业竞争力会越来越强！

2.4、Node.js 环境的安装

4.1 如果希望通过 Node.js 来运行 Javascript 代码，则必须在计算机上安装 Node.js 环境才行。

安装包可以从 Node.js 的官网首页直接下载，进入到 Node.js 的官网首页

(<https://nodejs.org/en/>)，点击绿色的按钮，下载所需的版本后，双击直接安装即可。



4.2 查看已安装的Node.js的版本号

打开终端，在终端输入命令 `node -v` 后，按下回车键，即可查看已安装的 Node.js 的版本号。

Windows 系统快速打开终端的方式：

使用快捷键（Windows徽标键 + R）打开运行面板，输入 `cmd` 后直接回车，即可打开终端。

2.5、在 Node.js 环境中执行 JavaScript 代码

- 打开终端
- 输入 node 要执行的js文件的路径

5.1 终端中的快捷键

在 Windows 的 powershell 或 cmd 终端中，我们可以通过如下快捷键，来提高终端的操作效率：

- 使用 `↑` 键，可以快速定位到上一次执行的命令
- 使用 `tab` 键，能够快速补全路径
- 使用 `esc` 键，能够快速清空当前已输入的命令
- 输入 `cls` 命令，可以清空终端

3、fs 文件系统模块

3.1、什么是 fs 文件系统模块

fs 模块是 Node.js 官方提供的、用来操作文件的模块。它提供了一系列的方法和属性，用来满足用户对文件的操作需求。

例如：

- `fs.readFile()` 方法，用来读取指定文件中的内容
- `fs.writeFile()` 方法，用来向指定的文件中写入内容

如果要在 JavaScript 代码中，使用 fs 模块来操作文件，则需要使用如下的方式先导入它：

```
1 const fs = require('fs')
```

3.2、读取指定文件中的内容

1. fs.readFile() 的语法格式

使用 fs.readFile() 方法，可以读取指定文件中的内容，语法格式如下：

```
1 fs.readFile(path[, options], callback)
```

参数解读：

- 参数1：必选参数，字符串，表示文件的路径。
- 参数2：可选参数，表示以什么编码格式来读取文件。
- 参数3：必选参数，文件读取完成后，通过回调函数拿到读取的结果。

2. 示例代码：

以 utf8 的编码格式，读取指定文件的内容，并打印 err 和 dataStr 的值：

```
1 const fs = require('fs')
2 fs.readFile('./files/11.txt', 'utf8', function(err, dataStr) {
3   console.log(err)
4   console.log('-----')
5   console.log(dataStr)
6 })
```

3. 判断文件是否读取成功

可以判断 err 对象是否为 null，从而知晓文件读取的结果：

```
1 const fs = require('fs')
2 fs.readFile('./files/1.txt', 'utf8', function(err, result) {
3   if (err) {
4     return console.log('文件读取失败！' + err.message)
5   }
6   console.log('文件读取成功，内容是：' + result)
7 })
```

```
1 //1 导入 fs 模块，来操作文件
2 const fs = require('fs')
3
```

```

4  //2 调用 fs.readFile() 方法读取文件
5  // 参数1: 读取文件的存放路径
6  // 参数2: 读取文件时候采用的编码格式, 一般默认指定 utf8
7  // 参数3: 回调函数, 拿到读取失败和成功的结果 err  dataStr
8  fs.readFile('./files/11.txt', 'utf8', function (err, dataStr) {
9      //3 打印失败的结果
10     // 如果读取成功, 则 err 的值为 null
11     // 如果读取失败, 则 err 的值为 错误对象, dataStr 的值为 undefined
12     console.log(err)
13     console.log('-----')
14     //4 打印成功的结果
15     console.log(dataStr)
16 })

```

3.3、向指定的文件中写入内容

1. fs.writeFile() 的语法格式

使用fs.writeFile()方法, 可以向指定的文件中写入内容, 语法格式如下:

```

1 fs.writeFile(file, data[, options], callback)

```

参数解读:

- 参数1: 必选参数, 需要指定一个文件路径的字符串, 表示文件的存放路径。
- 参数2: 必选参数, 表示要写入的内容。
- 参数3: 可选参数, 表示以什么格式写入文件内容, 默认值是 utf8。
- 参数4: 必选参数, 文件写入完成后的回调函数。

2. fs.writeFile() 的示例代码

向指定的文件路径中, 写入文件内容:

```

1 const fs = require('fs')
2 fs.writeFile('./files/2.txt', 'Hello Node.js!', function(err) {
3     console.log(err)
4 })

```

3. 判断文件是否写入成功

可以判断 err 对象是否为 null, 从而知晓文件写入的结果:

```

1 const fs = require('fs')
2 fs.writeFile('F:/files/2.txt', 'Hello Node.js!', function(err) {
3   if (err) {
4     return console.log('文件写入失败! ' + err.message)
5   }
6   console.log('文件写入成功! ')
7 })

```

```

1 const fs = require('fs')
2
3 //2 fs.writeFile()方法
4 // 参数1: 路径
5 // 参数2: 内容
6 // 参数3: 字符编码
7 // 参数4: 回调函数
8
9 fs.writeFile('./files/2.txt', 'abcdefg', 'utf8', function (err) {
10   // console.log(err);
11   //写入成功: err等于null
12   //写入失败: err等于一个错误对象
13   if (err) {
14     return console.log('文件写入失败' + err.message);
15   }
16   console.log('文件写入成功');
17 })

```

3.4、练习 - 考试成绩整理

使用 fs 文件系统模块，将素材目录下成绩.txt文件中的考试数据，整理到成绩-ok.txt文件中。
整理前，成绩.txt文件中的数据格式如下：

```

1 小红=99 小白=100 小黄=70 小黑=66 小绿=88

```

整理完成之后，希望得到的成绩-ok.txt文件中的数据格式如下：

```
1 小红: 99
2 小白: 100
3 小黄: 70
4 小黑: 66
5 小绿: 88
```

核心实现步骤:

- 导入需要的 fs 文件系统模块
- 使用 fs.readFile() 方法, 读取素材目录下的 成绩.txt 文件
- 判断文件是否读取失败
- 文件读取成功后, 处理成绩数据
- 将处理完成的成绩数据, 调用 fs.writeFile() 方法, 写入到新文件 成绩-ok.txt 中

```
1 // 1. 导入 fs 模块
2 const fs = require('fs')
3
4 // 2. 调用 fs.readFile() 读取文件的内容
5 fs.readFile('../素材/成绩.txt', 'utf8', function (err, dataStr) {
6     // 3. 判断是否读取成功
7     if (err) {
8         return console.log('读取文件失败! ' + err.message)
9     }
10    // console.log('读取文件成功! ' + dataStr)
11
12    // 4.1 先把成绩的数据, 按照空格进行分割
13    const arrOld = dataStr.split(' ')
14    // 4.2 循环分割后的数组, 对每一项数据, 进行字符串的替换操作
15    const arrNew = []
16    arrOld.forEach(item => {
17        arrNew.push(item.replace('=', ':'))
18    })
19    // 4.3 把新数组中的每一项, 进行合并, 得到一个新的字符串
20    const newStr = arrNew.join('\r\n');
21    console.log(newStr);
22
23    // 5. 调用 fs.writeFile() 方法, 把处理完毕的成绩, 写入到新文件中
24    fs.writeFile('./files/成绩-ok.txt', newStr, function (err) {
25        if (err) {
26            return console.log('写入文件失败! ' + err.message)
```



```
27         }
28         console.log('成绩写入成功! ')
29     })
30 }
```

The screenshot shows a VS Code interface with a terminal window. The file explorer on the left displays the project structure: a 'code' folder containing 'clock' and 'files' subfolders. The 'files' folder contains '1.txt', '2.txt', '成绩-ok.txt', '为学.txt', and a JavaScript file '01.读取文件.js'. The terminal window has tabs for '输出' (Output), '终端' (Terminal), '调试控制台' (Debug Console), and '问题' (Issues). The '终端' tab is active, showing the command 'E:\Nodejs\Nodejs案例\day1\code>node 03.成绩整理.js' and its output: '小红:99', '小白:100', '小黄:70', '小黑:66', '小绿:88', and '成绩写入成功!'. The terminal prompt is 'E:\Nodejs\Nodejs案例\day1\code>'.

3.5、fs模块-修改文件名

语法:

```
fs.rename(oldPath, newPath, callback)
```

```
fs.renameSync(oldPath, newPath)
```

```
1 fs.renameSync('./resouce/1.txt', './resouce/2.md');
```

3.6、fs 模块 - 路径动态拼接的问题

在使用 fs 模块操作文件时，如果提供的操作路径是以 ./ 或 ../ 开头的相对路径时，很容易出现路径动态拼接错误的问题。

原因：代码在运行的时候，会以执行 node 命令时所处的目录，动态拼接出被操作文件的完整路径。

解决方案：

- 在使用 `fs` 模块操作文件时，直接提供完整的路径，不要提供 `./` 或 `../` 开头的相对路径，从而防止路径动态拼接的问题。
- `__dirname`(双下划线)

```
1  const fs = require('fs')
2
3  //1 出现路径问题是因为出现 ./ 或 ../开头的相对路径
4  //提供一个完整的路径可以解决问题
5  // fs.readFile('./files/1.txt', 'utf8', function (err, dataStr) {
6  //      if (err) {
7  //          console.log('读取失败' + err.message);
8  //      }
```

```

9 // console.log('读取成功' + dataStr);
10 // })
11
12 //2 绝对路径：移植性很差，不利于维护
13 // fs.readFile('E:\\Nodejs\\Nodejs案例\\day1\\code\\files\\1.txt', 'utf8', function
14 // (err, dataStr) {
15 //     if (err) {
16 //         return console.log('读取失败' + err.message);
17 //     }
18 //     console.log('读取成功' + dataStr);
19 // })
20
21 //3 __dirname(双下划线) 表示当前文件所处的目录
22 // console.log(__dirname);
23 fs.readFile(__dirname + '/files/1.txt', 'utf8', function (err, dataStr) {
24     if (err) {
25         return console.log('读取失败' + err.message);
26     }
27     console.log('读取成功' + dataStr);
28 })

```

4、path 路径模块

4.1、什么是 path 路径模块

path 模块是 Node.js 官方提供的、用来处理路径的模块。它提供了一系列的方法和属性，用来满足用户对路径的处理需求。

例如：

- path.join() 方法，用来将多个路径片段拼接成一个完整的路径字符串
- path.basename() 方法，用来从路径字符串中，将文件名解析出来

如果要在 JavaScript 代码中，使用 path 模块来处理路径，则需要使用如下的方式先导入它：

```
1 const path = require('path')
```

4.2、路径拼接

```

1 const path = require('path');
2 console.log(path.dirname(__filename)); //F:\01.H220302全栈\03day.Nodejs

```

```
3 console.log(path.dirname(__dirname)); //F:\01.H220302全栈
4
5 console.log(path.dirname(path.dirname(path.dirname('c:/a/b/d.txt'))));
6
7 console.log(path.basename(__filename)); //单纯的打印文件名，不包含路径
8 console.log(path.extname(__filename)); //扩展名
```

1. path.join() 的语法格式

使用 path.join() 方法，可以把多个路径片段拼接为完整的路径字符串，语法格式如下：

```
1 path.join([...paths])
```

参数解读：

- ...paths <string> 路径片段的序列
- 返回值: <string>

2. path.join() 的代码示例

使用 path.join() 方法，可以把多个路径片段拼接为完整的路径字符串：

```
JS 06.path.join()方法.js ×
Nodejs案例 > day1 > code > JS 06.path.join()方法.js > ...
1  const path = require('path');
2  const fs = require('fs');
3
4  //注意： ../会抵消前面的路径
5  // const pathStr = path.join('/a', '/b/c', '../', './d', 'e');
6  // console.log(pathStr); // 结果: \a\b\d\e
7
8  //fs.readFile(__dirname + '/files/1.txt');
9
10 fs.readFile(path.join(__dirname, '/files/1.txt'), 'utf8', function (err,
11     if (err) {
12         return console.log(err.message);
13     }
14     console.log(data);
15 })

输出 终端 调试控制台 问题
cmd + 展开 收起 删除 上一步 下一步
E:\Nodejs\Nodejs案例\day1\code>node "06.path.join()方法.js"
读取成功123!!!
E:\Nodejs\Nodejs案例\day1\code>
```

注意：今后凡是涉及到路径拼接的操作，都要使用 path.join() 方法进行处理。不要直接使用 + 进行字符串的拼接。

4.3、获取路径中的文件名

1. path.basename() 的语法格式

使用 path.basename() 方法，可以获取路径中的最后一部分，经常通过这个方法获取路径中的文件名，语法格式如下：

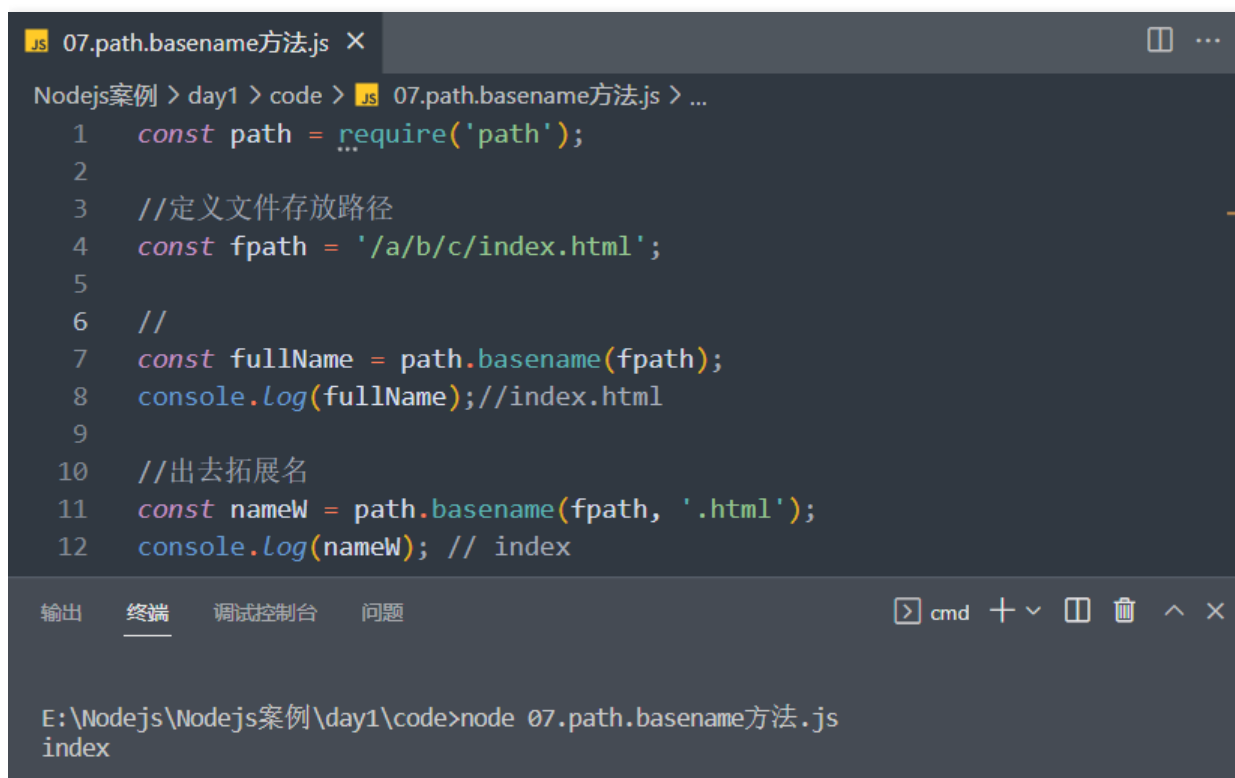
```
1 path.basename(path[, ext])
```

参数解读：

- path <string> 必选参数，表示一个路径的字符串
- ext <string> 可选参数，表示文件扩展名
- 返回: <string> 表示路径中的最后一部分

2. path.basename() 的代码示例

使用 path.basename() 方法，可以从一个文件路径中，获取到文件的名称部分：



```
JS 07.path.basename方法.js X
Nodejs案例 > day1 > code > JS 07.path.basename方法.js > ...
1  const path = require('path');
2
3  //定义文件存放路径
4  const fpath = '/a/b/c/index.html';
5
6  //
7  const fullName = path.basename(fpath);
8  console.log(fullName); //index.html
9
10 //出去拓展名
11 const nameW = path.basename(fpath, '.html');
12 console.log(nameW); // index

输出 终端 调试控制台 问题 cmd +v  删除 上一步 下一步
E:\Nodejs\Nodejs案例\day1\code>node 07.path.basename方法.js
index
```

4.4、获取路径中的文件扩展名

1. path.extname() 的语法格式

使用path.extname()方法，可以获取路径中的扩展名部分，语法格式如下：

```
1 path.extname(path)
```

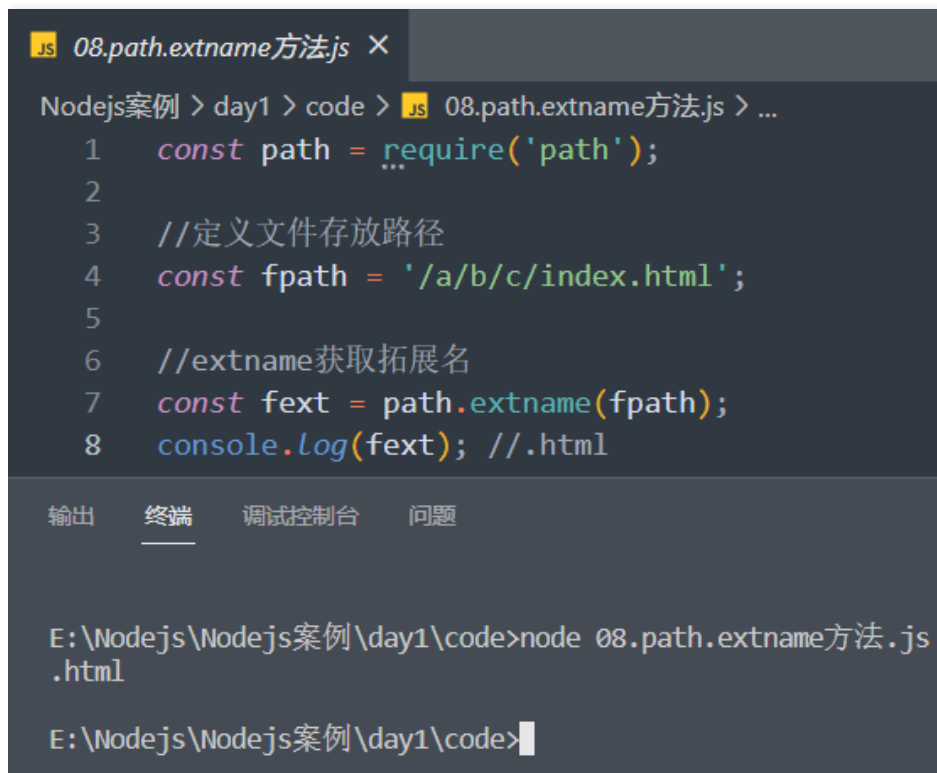
参数解读：

- path <string> 必选参数，表示一个路径的字符串

- 返回: <string> 返回得到的扩展名字符串

2. path.extname() 的代码示例

使用 path.extname() 方法，可以获取路径中的扩展名部分：



```
JS 08.path.extname方法.js X
Nodejs案例 > day1 > code > JS 08.path.extname方法.js > ...
1  const path = require('path');
2
3  //定义文件存放路径
4  const fpath = '/a/b/c/index.html';
5
6  //extname获取拓展名
7  const fext = path.extname(fpath);
8  console.log(fext); //.html

输出 终端 调试控制台 问题

E:\Nodejs\Nodejs案例\day1\code>node 08.path.extname方法.js
.html

E:\Nodejs\Nodejs案例\day1\code>
```

4.5、综合案例 - 时钟案例

1. 案例要实现的功能

将素材目录下的 index.html 页面，拆分成三个文件，分别是：

- index.css
- index.js
- index.html

并且将拆分出来的 3 个文件，存放到 clock 目录中。

2. 案例的实现步骤

- 创建两个正则表达式，分别用来匹配 <style> 和 <script> 标签
- 使用 fs 模块，读取需要被处理的 HTML 文件
- 自定义 resolveCSS 方法，来写入 index.css 样式文件
- 自定义 resolveJS 方法，来写入 index.js 脚本文件
- 自定义 resolveHTML 方法，来写入 index.html 文件

3. 步骤1 - 导入需要的模块并创建正则表达式

```

1 // 1.1 导入 fs 文件系统模块
2 const fs = require('fs')
3 // 1.2 导入 path 路径处理模块
4 const path = require('path')
5
6 // 1.3 匹配 <style></style> 标签的正则
7 // 其中 \s 表示空白字符; \S 表示非空白字符; * 表示匹配任意次
8 const regStyle = /<style>[\s\S]*</style>/
9 // 1.4 匹配 <script></script> 标签的正则
10 const regScript = /<script>[\s\S]*</script>/

```

4. 步骤2 - 使用 fs 模块读取需要被处理的 html 文件

```

1 // 2.1 读取需要被处理的 HTML 文件
2 fs.readFile(path.join(__dirname, '../素材/index.html'), 'utf8', (err, dataStr) => {
3   // 2.2 读取 HTML 文件失败
4   if (err) return console.log('读取HTML文件失败!' + err.message)
5
6   // 2.3 读取 HTML 文件成功后, 调用对应的方法, 拆解出 css、js 和 html 文件
7   resolveCSS(dataStr)
8   resolveJS(dataStr)
9   resolveHTML(dataStr)
10 })

```

5. 步骤3 – 自定义 resolveCSS 方法

```

1 // 3.1 处理 css 样式
2 function resolveCSS(htmlStr) {
3   // 3.2 使用正则提取页面中的 <style></style> 标签
4   const r1 = regStyle.exec(htmlStr)
5   // 3.3 将提取出来的样式字符串, 做进一步的处理
6   const newCSS = r1[0].replace('<style>', '').replace('</style>', '')
7   // 3.4 将提取出来的 css 样式, 写入到 index.css 文件中
8   fs.writeFile(path.join(__dirname, './clock/index.css'), newCSS, err => {
9     if (err) return console.log('写入 CSS 样式失败!' + err.message)
10    console.log('写入 CSS 样式成功!')
11  })
12 }

```

6. 步骤4 – 自定义 resolveJS 方法

```

1 // 4.1 处理 js 脚本
2 function resolveJS(htmlStr) {
3   // 4.2 使用正则提取页面中的 <script></script> 标签
4   const r2 = regScript.exec(htmlStr)
5   // 4.3 将提取出来的脚本字符串，做进一步的处理
6   const newJS = r2[0].replace('<script>', '').replace('</script>', '')
7   // 4.4 将提取出来的 js 脚本，写入到 index.js 文件中
8   fs.writeFile(path.join(__dirname, './clock/index.js'), newJS, err => {
9     if (err) return console.log('写入 JavaScript 脚本失败! ' + err.message)
10    console.log('写入 JS 脚本成功! ')
11  })
12 }

```

7. 步骤5 – 自定义 resolveHTML 方法

```

1 // 5. 处理 html 文件
2 function resolveHTML(htmlStr) {
3   // 5.1 使用字符串的 replace 方法，把内嵌的 <style> 和 <script> 标签，替换为外联的 <link> 和 <script> 标签
4   const newHTML = htmlStr
5     .replace(regStyle, '<link rel="stylesheet" href="./index.css"/>')
6     .replace(regScript, '<script src="./index.js"></script>')
7   // 5.2 将替换完成之后的 html 代码，写入到 index.html 文件中
8   fs.writeFile(path.join(__dirname, './clock/index.html'), newHTML, err => {
9     if (err) return console.log('写入 HTML 文件失败! ' + err.message)
10    console.log('写入 HTML 页面成功! ')
11  })
12 }

```

8. 案例的两个注意点

- fs.writeFile() 方法只能用来创建文件，不能用来创建路径
- 重复调用 fs.writeFile() 写入同一个文件，新写入的内容会覆盖之前的旧内容

```

1 //0 导入fs path模块
2 const fs = require('fs');
3 const path = require('path');
4
5 //1 定义正则表达式，分别匹配<style></style>和<script></script>标签
6 const regStyle = /<style>[\s\S]*</style>/;
7 const regScript = /<script>[\s\S]*</script>/;
8
9 //2 调用 fs.readFile() 方法读取文件

```

```
10 fs.readFile(path.join(__dirname, '../素材/index.html'), 'utf8', function (err, dataStr)
11 {
12     // 2.1 读取 HTML 文件失败
13     if (err) return console.log('读取HTML文件失败!' + err.message);
14     // 2.2 读取文件成功后，调用对应的三个方法，分别拆解出 css, js, html 文件
15     resolveCSS(dataStr);
16     resolveJS(dataStr);
17     resolveHTML(dataStr);
18 })
19 //注意：fs.writeFile()方法只能创建文件，不能创建路径
20 // 重复调用fs.write()方法写入一个文件，新的文件内容会覆盖旧的文件内容
21
22 //3 定义处理CSS的方法
23 function resolveCSS(htmlStr) {
24     //3.1 正则提取内容
25     const r1 = regStyle.exec(htmlStr);
26     //3.2 将提取出来的样式字符串，进行replace替换操作
27     const newCSS = r1[0].replace('<style>', '').replace('</style>', '');
28     //3.3 使用fs.writeFile()方法，将提取的样式写入clock目录中的index.css文件中
29     fs.writeFile(path.join(__dirname, './clock/index.css'), newCSS, function (err) {
30         if (err) return console.log('写入CSS样式失败 ' + err.message);
31         console.log('写入CSS成功! ');
32     })
33 }
34
35 //4 定义处理JS的方法
36 function resolveJS(htmlStr) {
37     const r2 = regScript.exec(htmlStr);
38
39     const newJS = r2[0].replace('<script>', '').replace('</script>', '');
40     fs.writeFile(path.join(__dirname, './clock/index.js'), newJS, function (err) {
41         if (err) return console.log('写入JS代码失败 ' + err.message);
42         console.log('写入JS成功! ');
43     })
44 }
45
46 //5 定义处理html的方法
47 function resolveHTML(htmlStr) {
48     //5.1 把内嵌的style和script标签替换为link标签
```



```

49     const newHTML = htmlStr.replace(regStyle, '<link rel="stylesheet"
      href="./index.css"/>').replace(regScript, '<script src="./index.js"></script>');
50     //5.2 写入index.html文件
51     fs.writeFile(path.join(__dirname, './clock/index.html'), newHTML, function (err) {
52         if (err) return console.log('写入失败 ' + err.message);
53         console.log('写入html成功 ');
54     })
55 }

```

JS 09.时钟案例.js ×

Nodejs案例 > day1 > code > JS 09.时钟案例.js > resolveCSS

```

1  //0 导入fs path模块
2  const fs = require('fs');
3  const path = require('path');
4
5  //1 定义正则表达式，分别匹配<style></style>和<script></script>标签
6  const regStyle = /<style>[\s\S]*</style>/;
7  const regScript = /<script>[\s\S]*</script>/;
8
9  //2 调用 fs.readFile() 方法读取文件
10 > fs.readFile(path.join(__dirname, '../素材/index.html'), 'utf8', function
17 })
18
19 //注意：fs.writeFile()方法只能创建文件，不能创建路径
20 // 重复调用fs.write()方法写入一个文件，新的文件内容会覆盖旧的文件内容
21
22 //3 定义处理CSS的方法
23 > function resolveCSS(htmlStr) { ...
33 }
34
35 //4 定义处理JS的方法
36 > function resolveJS(htmlStr) { ...
44 }
45
46 //5 定义处理html的方法
47 > function resolveHTML(htmlStr) { ...
55 }

```

输出 终端 调试控制台 问题

cmd + 图标

```

E:\Nodejs\Nodejs案例\day1\code>node 09.时钟案例.js
写入html成功
写入css成功!
写入JS成功!

```

5、OS模块

```

1  const os = require('os');
2  console.log(os.EOL);//获取当前操作系统的换行符
3
4  console.log(os.cpus());//获取cpu信息
5  console.log(os.freemem());
6  console.log(os.platform());
7  console.log(os.hostname());

```

```

1  console.log(global);//window
2  console.log(globalThis);//把global 和 window 合二为一
3  console.log(global === globalThis);
4
5  console.log(__filename);//获取的是当前文件的完整磁盘路径
6  console.log(__dirname);//directory 当前文件所在的目录

```

6、web服务器创建

```

1  工程目录:
2  08_web:
3    ----asset:
4      +-----a.html
5      +-----b.html
6      +-----index.html
7    ----app.js
8    -----
9  const http = require('http')
10 const fs = require('fs')
11
12 const app = http.createServer((req, res) => {
13   console.log(req.url);
14   let url = req.url;
15   if (url === '/favicon.ico') {
16     res.end('/favicon.ico');
17     return;

```

```

18     }
19     if (url === '/') url = '/index';
20     console.log(url);
21     fs.readFile(`./asset${url}.html`, {}, (err, data) => {
22         res.setHeader('Content-Type', 'text/html; charset="utf-8"')
23         if (err) {
24             res.write('你访问的页面不存在-404')
25         } else {
26             res.write(data)
27         }
28         res.end();
29     })
30
31 })
32
33 app.listen(3000, () => {
34     console.log('服务器启动了 at: http://localhost:3000');
35 })

```

7、第三方模块使用

查看npm官网

eg:

```

1 // md5 加密
2 const md5 = require('md5')
3
4 let passwd = '123456';
5 console.log(md5(passwd));
6 // e10adc3949ba59abbe56e057f20f883e

```

mysql:

```

1 const http = require('http');
2 const mysql = require('mysql');
3
4 const app = http.createServer((req, res) => {
5     res.setHeader('content-type', 'text/html; charset="utf8"')
6     let connection = mysql.createConnection({

```

```
7      host: 'localhost',
8      user: 'root',
9      password: '123456',
10     database: 'db_01'
11   })
12   connection.connect();
13
14   const sql = 'select avg(salary) avgсал from yg group by deptid';
15
16   connection.query(sql, (err, results) => {
17     if (err) throw err;
18     res.end(JSON.stringify(results))
19   })
20
21 })
22
23 app.listen(81, '192.168.31.30', () => {
24   console.log('1111');
25 })
```

8、自定义模块

